



A SHIFT OF ACCENT IN INTRODUCTORY OBJECT ORIENTED PROGRAMMING COURSES

Eugeni Gentchev*, Claudia Roda**

* The American University of Paris, Computer Science Department
148 rue de Grenelle, 75007 Paris, France, egentchev@aup.fr

** The American University of Paris, Computer Science Department
148 rue de Grenelle, 75007 Paris, France, claudia.roda@aup.fr

Summary. With the advent of modern Application Life Management (ALM) methodologies and tools, the focus is shifting from the design and implementation of components to the analysis and modeling of overall applications. As model driven architectures become available, programming as we know it, will disappear, to be replaced by the development of models and meta-models that will be automatically or semi-automatically compiled into target platforms. We believe that this is more than just a change in paradigm, it is instead a change in abstraction level, just as the shift from assembler-like languages to high level programming languages, was. In this paper we will concentrate on two consequences of the above change of abstraction level. First while still teaching object oriented programming, we discuss how it is time to shift the accent from the construction phase of the application development, to the design and modeling phase of the ALM. Tools are available to easily switch between modeling using UML and programming using, for example, Java or .NET.

Keywords: design, Java, modeling, programming, teaching

INTRODUCTION

Recently we have witnessed a fast progress in the methodologies and in the facilities provided by software development tools for Object Oriented Programming (OOP). This progress changes the approach to software development in industry, and we believe that such change should be reflected in education.

During the time of procedural programming, the emphasis was on the creation of algorithms and it was important to teach students first how to develop algorithms and then how to program them. When Object Oriented methodologies were introduced, the emphasis in education was placed on making sure that students would understand the Object Oriented concepts and that they could program them.

On the purely object oriented programming side, there are a lot of advances that make OOP more efficient but also easier. For example, learning the syntax of a programming language is becoming less critical due to the availability of "syntax check as you type", code completion, design patterns, wizards, etc.

We begin by introducing the results of a small study indicating that, in object-oriented event-driven applications, short and less complicated statements form the bulk of the code, and that less simple statements are used in semi-standard manner. On the basis of these findings we argue that the complexity of software implementation does not reside anymore in program development but rather in system design. Then we discuss how this

shift of focus in modern software development could be addressed when teaching introductory computer science courses.

Teaching control structures in Object oriented programming is less important?

As a starting point we decided to analyze a professionally developed Web based Java version of the well known Pet store application (Java Pet Store Software 1.3.2: Copyright 2003 Sun Microsystems, Inc. All Rights Reserved.) in order to determine the importance of different Java language constructs and the typical circumstances of their use.

The Sun Microsystems developed Java Pet store application was originally developed to demonstrate the capabilities of the J2EE 1.3 platform. It consists of 283 java files, 369 classes and 9811 lines of Java code.

Our first analysis of the Java code concentrated on evaluating which statements were most often used and it is summarized in table 1.

Statement	Occurrences	Percentage	in files
assignment	3274	33%	207
if	621	6.3%	118
if..else	196	1.2%	65
for	104	1.06%	29
while	22	0.22%	14
do...while	8	0.08%	2

Table 1 – Frequency of main Java statements in the Pet Store application

The results confirmed our hypotheses that in object-oriented event-driven applications short and less complicated statements form the bulk of the code. Amazingly enough **while** loop is only used 22 times and it is used in only 14 files. Whilst this obviously does not imply that less used statements are un-necessary, it indicates a clear shift of the locus of complexity in modern software.

We took a more detailed look at the circumstances of the use of control structures in order to verify our hypothesis that *standard* use of main control structures actually covered a large percentage of the code. The results of this analysis are reported in Table 2.

Statement	Occurrences	Percent	In files	Percent
if (var != null) {	269	43% of if	78	66%
if (var == null) {	151	24% of if	67	57%
for (int i=0; i<orders.length;i++) {	29	28% of for	9	31%
for (int loop=0;loop<list.getLength();loop++){	38	36% of for	7	24%
for (int row=0;row<row.count;row++){	4	4% of for	2	
while (it.hasNext()) {	9	37% of while	7	50%
while(enm.hasMoreElements()) {	7	29% of while	4	28%
while (it !=null) && it.hasNext()) {	3	12% of while	3	21%

Table 2 - Circumstances of use of some the control structures in the Pet Store application

This results indicate that even in the cases when less simple statements are used, the complexity of software implementation does not reside anymore in program development. Furthermore, current development tools make it even easier offering "syntax check as you type", code completion, etc. One of the most widely used Java development tool, Eclipse, offers "lazy programming" features for using loops. For example if we write a method with a String parameter:

```
public void method1 (String myString) {  
    for  
}
```

and after the keyword **for** we invoke the code completion feature by typing CTRL-SPACE, Eclipse will show pop-up with the following options:

```
“for - iterate over array  
  for - iterate over array with temporary variable  
  for - iterate over collection”.
```

If we choose the first one, Eclipse will generate

```
public void method1 (String[] myString) {  
    for (int i = 0; i < myString.length; i++) {  
    }  
}
```

Wasn't this the most widely use of **for** loop in the Java Pet Store Application?

If we choose the second option, Eclipse will iterate the loop using temporary variable:

```
for (int i = 0; i < myString.length; i++) {  
    String string = myString [i];  
}
```

If we choose the third option, Eclipse will generate iteration over collection.

Teaching analysis and Object Oriented design is more important.

Several authors [1,2,3] use UML and similar visual modeling (e.g. BlueJ) in introductory programming courses to facilitate the understanding of objects and object oriented concepts. Bennedsen [1] observed that "teaching how to convert a UML model of the system into a working Java program in a very systematic way gives the students a path to follow when they create programs – create a UML model of the program and then convert it into a Java program. It is often easier for these students to make a model of the system in UML and then convert it to a program than directly programming it.[1, p5]

The authors of BlueJ [2], a Java integrated development environment specifically designed for education, also believe that teaching objects is the most important issue and it should be done first (the pedagogical design patterns "Early bird" [5] states that we need to hide as many details as possible in the beginning, just focusing on what is absolutely needed to do the job).

BlueJ presents on screen a graphical overview of a project structure in the form of a UML-like class diagram. It then allows the interactive creation of objects from any given class in a software project. Once an object has been created, it becomes visible to the user and any of its public methods can be interactively invoked by selecting them from a pop-up menu. Parameters and method results are entered and presented through dialogue windows.

Amongst his Guidelines for Teaching Object Orientation with Java Michael Kölling [2] recommends to *show program structure* and he states that “In discussing object-oriented programming, the classes and their relationships are a central issue. [1] Yet in most development environments, the internal program structure is not visible. It is very hard to discuss issues in a completely abstract way. Visualizing the class structure is crucial for students to develop an understanding of the important concepts. If programming environments do not provide functionality to display that structure, teachers must take great care to present visual representations by other means as a basis for discussion. BlueJ facilitates this discussion by automatically computing and displaying the class diagram” [p. 3].

We fully agree that “the structure of the application is crucial to the quality of a solution” [ibid, p.3], but this is a system design problem and not programming problem. Using BlueJ to visualize the class structure will definitely help students to understand important OO concepts playing with carefully designed examples. What about teaching students to solve problems?

We think the time has come to move farther in the direction of problem analysis and application design using UML and UML-based integrated development environment. First of all, the analysis and design phases precede the construction phase of an application and, at this level, it is indispensable to understand the class structure and interaction between objects.

We believe that teaching the basic concepts of OO analysis and design, and OO programming, should be done in the same course. There are two possible scenarios for achieving this objective: start with analysis and design and then move into implementing the design programmatically, or moving “in parallel” with analysis-design-programming. Lets take a look at how this approach not only supports, but reinforces the above mentioned Guidelines for Teaching Object Orientation with Java.

"Guideline 4: *Use large projects*" [2, p2] states that “a serious problem with teaching object orientation to beginners is that object-oriented programs have some syntactic overhead. The overhead is roughly proportional to the number of classes (or constant for each class). This is a problem only because the reason for having this overhead is not immediately apparent to beginners. This problem seems worse the shorter the program is. If the intention is to execute a single line of code, then the syntactical overhead is, relatively speaking, large. It appears more acceptable in longer programs. Some of the

benefits of object orientation only become apparent in larger programs. ...Students should see sensible examples from the start. In particular they should see that a Java application consists of a set of cooperating classes. Thus, we should show them “large” (in student terms) examples, which consist of several classes with a sensible number of methods.” [ibid, p.2]

Perfect! We need to start with “large” projects. For large projects it makes more sense to start with the analysis and design and express the design using UML diagrams. Then we can use modeling tools like Rational XDE for Java or JBuilder Together edition to generate the Java code. In this way the students can practice the reading of code first and after they feel comfortable they can inverse the process by first writing the code and then use the tool to generate the class diagrams. However, one problem we see with current professional modeling tools is that they are very heavy and quite difficult to learn. This could be overcome by developing generation feature for educational tool like BlueJ. This methodology would also meet the growing trend in industrial software development of using visual modeling. In industrial environments visual modeling significantly improves the quality of software development whilst lowering its cost.

CONCLUSIONS

The proposed shift of accent in introductory object oriented programming from teaching the programming language constructs into teaching object oriented concepts starting with analysis and visual design and then moving into the object oriented programming, has in our view double benefits. First at all, it will facilitate the learning of object oriented programming, and second of all it will help the students master another level of abstraction they’ll need as the software development moves towards model-driven architecture [6].

REFERENCES:

1. Jens Bennedsen, Department of Computer Science, University of Aarhus, Denmark, jbb@daimi.au.dk : Teaching Java programming to media students with a liberal arts background
 2. Michael Kölling, School of Network Computing, Monash University, mik@monash.edu.au, John Rosenberg, Faculty of Information Technology
 3. Monash University, johnr@infotech.monash.edu.au: Guidelines for Teaching Object Orientation with Java
 4. David Barnes, Michael Kolling: “*Objects First with Java A Practical Introduction using BlueJ*”, Pearson Education, 2005
 5. D.S. Malik: *Java Programming From Program Analysis to Program Design*, Thomson, Boston, 2006
 6. Bergin, Joe: Fourteen Pedagogical patterns, csis.pacs.edu/~bergin/PedPat1.3.html
 7. S.JMellor, K.Scott, A.Uhl, D.Weise: *MDA Distilled Principles of Model-Driven Architecture*, Addison-Wesley, 2004
-

